THE UNIVERSITY OF MELBOURNE

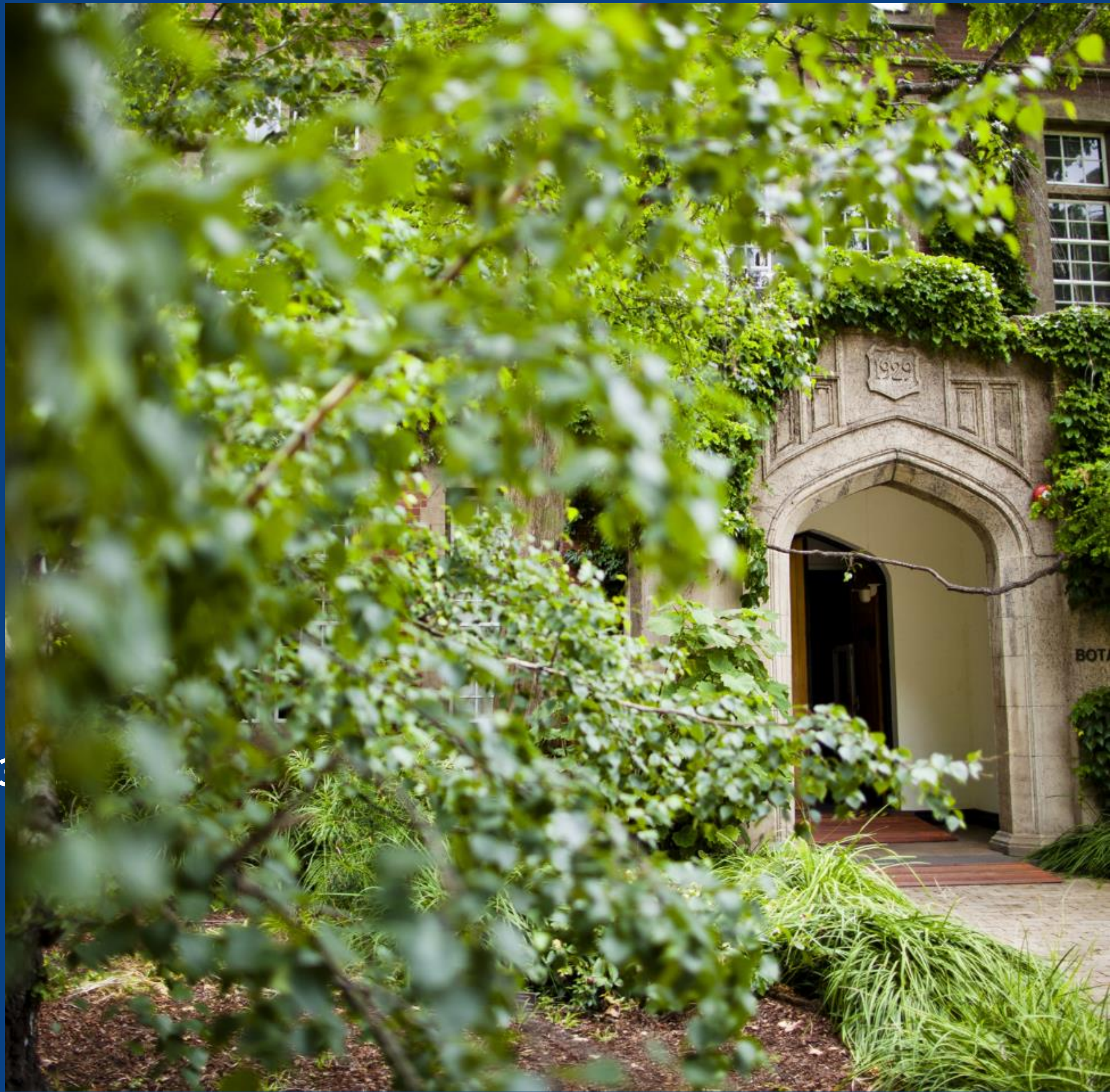
COMP 90048
Declarative Programming
Workshop 5 (week6)

2019 semester 1

by Wendy Zeng

Tutorial : Tue 18:15 - 19:15 221 Bouverie St, room B113

Wed 17:15 - 18:15 201 Bouverie St, room B132



Outline

1. Function Composition
2. Partial Application and Currying
3. More on Higher Order Functions
4. Code Quality Assessment for Proj1

1. Function Composition

(.) :: (b -> c) -> (a->b) -> a -> c

$\underbrace{\text{(b -> c)}}_{\text{function f}}$ $\underbrace{\text{(a->b)}}_{\text{function g}}$

function f function g

f (g x) = (f . g) x

func x = f1 (f2 (f3 (f4 x)))

↓

func x = (f1 . f2 . f3 . f4) x

↓

func = f1 . f2 . f3 . f4

- Example:

```
tree_sort :: (Ord a) => [a] -> [a]
tree_sort lst = tree_to_list (list_to_tree2 lst)
```

↓

```
tree_sort = tree_to_list . list_to_tree
```

2. Partial Application and Currying

- Partial Application:
  - Each function declared is "specialized" in doing one job at a time, by supplying one or more but not all of its arguments

- Currying:
  - For n-argument function: consider as a sequence of one-argument function, which "consumes" one argument at a time and return the next function for the rest of the computation
  - Evaluation in steps: Haskell internal intermediate function

- Example: ternary add
- ternary_add :: Int -> Int -> Int -> Int
- ternary_add a b c = a + b + c

= ((ternary_add a) b) c

2. Partial Application and Currying

Assume the polymorphic definition of the binary search tree datatype given in Section 06 of the lecture slides.

```
data Tree k v = Leaf | Node k v (Tree k v) (Tree k v)
```

Consider the function defined by

```
treeMapVal _ Leaf = Leaf
treeMapVal f (Node k v l r) = (Node k (f v) (treeMapVal f l) (treeMapVal f r))
```

(You can consider it the tree analogue of map, as regards the values stored in the binary search tree.)

Which one of the following most concisely and correctly gives the type of

```
treeMapVal (\n -> n /= 0)
```

- *The term is in error, because treeMapVal is given insufficient arguments.* — partial application
- Tree k v -> Tree k v — output tree has value of type Bool
- Num t => (t -> Bool) -> Tree k t -> Tree k Bool — (t -> Bool) has been supplied as (\n -> n/= 0)
- Num t => Tree k t -> Tree k Bool
- (Ord k, Num t) => Tree k t -> Tree k Bool — no comparison needed

2. Partial Application and Currying

- More Examples:
  - map length

    [[a]] -> [Int]
  - map (+3)

    Num a => [a] -> [a]
  - zip [True, False, False]

    [a] -> [(Bool, a)]
  - flip filter "hello"

    (Char -> Bool) -> [Char]
  - (. length)

    (Int -> b) -> [a] -> b

3. More on Higher Order Functions

- map :: (a -> b) -> [a] -> [b]
- filter :: (a -> Bool) -> [a] -> [a]
- foldl :: (b -> a -> b) -> b -> [a] -> b
- foldr :: (a -> b -> b) -> b -> [a] -> b
- zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]

- concatMap :: (a -> [b]) -> [a] -> [b] and how it's defined:
    - concatMap f list = foldr (++) [] (map f list)

      or using function composition
      and point-free style

    - concatMap = foldr ((++) . f) []

3. More on Higher Order Functions

- Q3: `linearEqn :: Num a => a -> a -> [a] -> [a]`

- **Solution 1:**
- `linearEqn m a = map (\x -> m*x+a)`

- **Solution 2:**
- `linearEqn m a = foldr ( (:) . (\x -> m*x+a) ) []`

- **Solution 3:**
- `linearEqn m a = foldl ( flip ( ( flip (++) ) . (\x -> [m*x+a] ) ) ) []`

3. More on Higher Order Functions

- Q4: allSqrt :: (Floating a, Ord a) => [a] -> [a]

- Solution 1:
- allSqrt = concatMap sqrtPM

- Solution 2:
- allSqrt = foldr ((++) . sqrtPM) []

- Solution 3:
- allSqrt lst = foldl (++) [] (map sqrtPM lst)
- Or using function composition:
- allSqrt lst = ((foldl (++) []) . (map sqrtPM)) lst

THE UNIVERSITY OF MELBOURNE

Thank you

wendy.zeng@unimelb.edu.au

By Wendy Zeng